形式仕様による開発試行
――仕様記述例の提示と実装実験の評価――

染　谷　　誠，古　村　哲　也

1. はじめに

形式的開発法とは，形式仕様言語を使って仕様書を記述し，その仕様書からその仕様を満たすコードを導出するための方法である．信頼性の高いソフトウェアの開発方法として大いに期待され，ソフトウェアの信頼性を追及するための，最終的な拠り所と目されている．

2. 開発試行の狙い

しかし，産業界では，形式的開発法は実験的試行の段階で，いまだ開発現場の標準的な方法にはなっていない．今回の開発試行の狙いは，やがて形式的開発法が開発現場の標準的方法のひとつとなるよう，その開発法がどのようなものであるかを具体的に知ってもらうことにある．

しかしながら，このようなささやかな実験で，形式的開発法の全貌を知ってもらうことは難しい．そのため今回の開発試行では，形式仕様の紹介とその一効用を実証することに狙いを絞った．

1） 仕様記述例の提示

開発試行の最初の作業項目は，仕様記述例を提示することである．その狙いは以下のとおりである．

① 形式仕様の紹介

形式的開発法の前提となる形式仕様とはどういうものか，それを共通 3 例題に即して具体的に提示する．既述の通り，仕様言語としてはＺ言語を取り上げる．Ｚ言語を使って，共通例題の仕様記述例を提示する．記述例を見ることが，形式仕様理解の早道だからである．

② 形式仕様に対する誤解を解消するために

産業界では，形式的開発法や形式仕様については，名前が広がっている割には，その実際はあまり知られていない．そのため，その名前から受ける印象からか，形式仕様は必要以上に難しいものと思われているようである．

形式仕様はなんらかの仕様言語を使って記述する．だからその仕様言語を知らない人には形式仕様は読めない．こういう事情があるため，どうしても，形式仕様は近づき難いとの印象を与えやすい．しかし，たとえばＺ仕様言語はプログラミング言語よりシンプルである．その文法を習得するだけなら，プログラミング言語より簡単である．

難しいのは決して仕様言語ではない．仕様を必要かつ十分に記述すること，これが難しいのだ．仕様言語はその難しい仕事を助けてくれる道具なのである．この事情を理解してもらうには，まず，身近な課題についてその仕様記述例を

実際に見てもらうことである．そうすることが，形式仕様に対する不当な誤解を解き，それを正しく理解してもらうための第一歩である．今回の開発試行の第一の狙い「仕様記述例の提示」には，このような意味もある．

2） 実装実験

作業項目の第二は，形式仕様の効用を検証する実装実験である．

① コードの分散開発が可能であることの実証

コード開発時の発注仕様に形式仕様を用いれば，コードの分散開発が可能であることを実装実験によって検証する．

コード開発の発注時には，必ず，開発すべきソフトウェアを規定するための文書，いわゆる仕様書がコード開発者（実装者）にわたされる．その仕様書には，開発すべきソフトウェアの仕様が必要かつ十分に，しかも自己充足的に書かれていなければ，期待どおりのコードは開発してもらえない．ところがそのような仕様を書くことは難しいため，実装者にわたされる仕様書は，仕様が必要十分に記述されていなかったり，自己充足的でなかったりすることが少なくない．これをカバーするため，実装者は仕様作成者の近傍で作業することが求められたりするようだ．いわば，コードの集中開発をおこなうわけである．

ところが，形式仕様を使えば，仕様を必要十分に，しかも自己充足的に記述できるので，コードが分散開発できるようになる．このことを実装実験によって検証する．

3. 仕様記述例提示のための技術的なテーマ

仕様記述例を提示するにあたり，提示する記述例の付加価値を高めるため，仕様記述上の技術的テーマを設定した．テーマは例題の性質に従って設定した．

共通例題は読みかた次第で大きな課題にもなり，小さな課題にもなる．今回の開発試行では，コードの開発が 1 人月以内を目安とし，例題文から開発対象範囲をどう切り出すかは，各例題の担当者に一任された．そこで，形式仕様による開発試行では，以下の技術的なテーマを対象範囲の切り出しの指針とした．

3.1 例題 1・例題 2 の仕様記述のための技術的テーマ

例題 1・例題 2 は典型的な事務処理の課題である．これら 2 例題の仕様記述では，以下の点に留意した．

1） 開発対象範囲の切り出しの理由づけ

開発試行の結果が，

・リアリティのあるサブシステムとなるよう，

・開発方法を議論するための素材となるよう，

開発対象の範囲の切り出しには，明確な理由付けをするようにした．

2） JSD による課題の整理

課題の整理では，JSD を参考とし，実体とその行動を洗い出した．但し，JSD に対して忠実ではない．

3） 実装環境からの独立

課題を整理し仕様を作成する段階では，実装環境になにを選ぶかはまったく意

識しなかった.

後述のように，実装ではRDBを採用した．実装を意識すれば，仕様と実装の差を少なくすることができる．RDBが関係と関係演算を基本としており，Z言語も集合を基本にしているので，特にその落差を少なくすることができる．そのようにしても，多くの場合，仕様も読みやすさをそれほど損なわれない．RDBによる実装ではこの点を利用することも考えてよい.

4) 記述水準の制限

Z言語の紹介を旨とし，記述の水準をZ言語が想定する記述水準に制限した．このため，計算資源に関する要求，基本データ型の実現の仕様などは記述の対象からはずした.

5) 構造化記述の例示

仕様言語Zは記述を構造化するための機構を豊富に備えている．仕様の構造化は巨大ソフトウェアの仕様記述には必須となる．小さな課題ながら，なるべく構造化の記述例を紹介するようにした.

3.2 例題3の仕様記述のための技術的テーマ

1) リフト運行のシミュレータの仕様

リフトの運行管理プログラムを実装する環境にはないので，リフト運行のシミュレータを開発することにした.

2) 状態遷移のトレースモデルの記述

リフトの運行管理プログラムやそのシミュレータの仕様には，状態遷移のトレースの性質を記述するのが自然である．ところが，仕様言語Zは，対象の状態とその状態の遷移を記述するための言語で，状態遷移のトレースを明示的に記述することは想定していない.

そこで，Z言語を少々拡張して，トレースモデルを記述することにした．トレースモデルの記述を主眼としたため，要求仕様としてはオーバースペックとなっている.

3) 仕様の構造化の記述例

仕様記述に際しては，なるべくZ言語の構造化のための記述機構を使うことを心がけた.巨大ソフトウェアの仕様記述のためには仕様の構造化は必須である.そこで，Z言語が構造化のためにどのような記述上の工夫をしているのかを紹介した．例題1・例題2にくらべて，多くの構造化記法を紹介することができた.

4) オブジェクト指向による仕様の構造化

仕様の構造化はオブジェクト指向を指針とした.

4. 実 装 実 験

開発試行の第二の作業項目は，形式仕様の効用を検証するための実装実験である.

4.1 検 証 命 題

実装実験で検証すべき命題は

「コード開発時の発注仕様を形式仕様にすれば，コードを分散開発できる」

である.

4.2 実 験 方 法

上記命題を検証するため，実装実験は以下のように進めた．

1） 協力企業と協同

実装実験は協力企業と協同して実施した．コードの実装はしばしば協力企業に発注する．したがって，形式仕様技術を開発現場の標準的技術のひとつとするためには，形式仕様からのコード開発を引き受けてくれる協力企業が必要になる．その意味でも，コードの分散開発による実装実験は，協力企業と協同して行うことが望ましい．

2） 実装者の育成

実装者の育成は基本的に協力企業に任せた．

3） 実装環境の選定

① 例題 1・例題 2：ACCESS（GUI とデータベースによる実装）

今日の事務処理分野のソフトウェア開発では，GUI とリレーショナルデータベースを採用することが多い．例題 1・例題 2 の実装には，この点を考慮して，ACCESS を選んだ．

② 例題 3：Java

例題 3 の仕様はオブジェクト指向にしたがって構造化した．そのため実装言語には Java か C＋＋を指定し，どちらを選ぶかは実装者に任せた．実装者は Java を選び，Applet として実装した．

4） 分散開発の実施

実装者と仕様作成者は離れた場所で作業した．

5） 電子メールによる交信

実装者と仕様作成者間での質疑応答は必ず電子メールでおこない，その記録を残すようにした．

4.3 実 験 結 果

各例題の実装実験の結果は以下の通りである．

1） 例題 1 の実装結果

① 分散開発はできたか

期待通りのコードを納期どおり開発することができた．したがってコードの分散開発を実現することができた．

② 交信内容の点検

例題 1 について，実装者と仕様作成者の間で交わされた，主な質疑応答は以下のとおりである．実装者はアクセスによる実現については習熟しているが，Z 言語からの実現は初めてとのことであった．

◆実装者が，実装の途中結果を送り，あるテーブルの実装方針が妥当かどうか，仕様作成者の判断を求めた．仕様作成者はそれを見て，もっとシンプルな構造で十分である旨，回答した．

◆実装者がスキーマ「注文 Item を製造元ごとに分類する」をどう判読すればよいかを質問した．同スキーマが判読できなかった理由はつぎの 2 点にあると思われる．

・そのスキーマで使われている「集合族の合併演算記号」の意味が分からなかった.

・表明による操作の定義方法に習熟していなかった.

この質疑応答によって，表明による操作の定義方法が理解できたようだ.

◆実装者は「注文」と「注文 Item」に関する RDB での実装方法について自身の案を提示したうえで，その妥当性を問うている．この質疑応答をとおして，実装者の Z 言語理解は急速に向上した.

2) 例題 2 の実装結果

① 分散開発はできたか

期待通りのコードを納期どおり開発できた．したがって例題 2 のコード開発も，分散開発ができた.

② 交信内容の点検

例題 2 について，実装者と仕様作成者の間で交わされた，主な質疑応答は以下のとおりである．例題 2 の実装者は例題 1 の実装者と同じである.

◆実装者は顧客台帳の実現方法について提案し，その可否を問うている．仕様作成者はもっと簡素な実装方法を示した.

3) 例題 3 の実装結果

① 分散開発はできたか

納期よりやや遅れたが，期待以上のコードが納入された．遅れの程度は 1 週程度であった.

この例題にも GUI を組み込むことが望ましいが，工数を押さえるため，GUI による実装は要求しなかった．ところが，納入されたコードは GUI を実装していた．その意味では，実質的には納期遅れとは言えない.

例題 3 の実装も，分散開発ができた.

② 交信内容の点検

例題 3 について，実装者と仕様作成者の間で交わされた，主な質疑応答は以下のとおりである.

◆実装者が，仕様中で使われているいくつかのスキーマ名について，それらが未定義のまま使用されていることを指摘した.

それら未定義のスキーマ名が生じたのは，仕様を書いている途中で何回か名前を変更したことが誘因となった．名前を変更するたび，古い名前を新しい名前に書き直したはずだが，徹底しきれなかったようだ.

この種の誤りは普通の注意力ではとてもカバーしきれない．だが，この種の誤りは型検査系を使えば直ちに検出できる．ただ，問題なのは，型検査系は英語版はあるが，日本語対応版がないことである.

日本語対応の型検査系を早急に開発すべきである.

◆実装者がプログラムの構造化について仕様作成者に相談している．はじめに，実装者は，例題 3 の仕様が，ある構造をもっていることを判読した．そのうえで，その構造をコードに反映させるべきだと判断．だが，Java でその構造をどう表現すればよいのかが分からない．ついてはその表現方

法を教えてほしい，という相談である．

実装者が判読した構造とは，実は，オブジェクト指向風の構造．したがって，その表現法は抽象データ型プログラミングをこころえていれば直ちに解決する．仕様作成者が疑似コードでその表現法を例示すると，実装者はその要点を直ちに理解した．

◆実装者と仕様作成者のあいだに，仕様理解について不整合があることが，コードの検収時に顕在化した．まず，仕様作成者が，その実装コードについて，「仕様に不適合な動きが見られる」と指摘したのに対して，実装者が，「仕様どおりである」と反駁したからである．

しかし，仕様作成者が，仕様書の該当箇所をあげ，プログラムがそこに記された要求を満たしていないことを指摘すると，実装者は仕様作成者の主張を認め，議論は直ちに収束した．仕様が曖昧さを許さないため，議論の正否がはっきりする．そのため議論の紛れる余地が少い．

◆前項に記した議論が契機となり，仕様作成者は仕様を改善することができた．仕様の第１版では，リフトの利用者の要求は無謬であることを前提にしていた．しかし，利用者のボタンの操作には当然ミスがあるため，その仕様は現実的ではない．

仕様作成者は，実装者と交わした議論がヒントになって，利用者の理不尽な要求を御破算にする仕様を盛り込むことができた．

4） 検　　証

実装実験の結果は，明らかに，検証例題「コード開発時の発注仕様を形式仕様にすれば，コードを分散開発できる」を肯定している．

実験から判明したことはそれだけではない．コードの分散開発を一層なめらかに推進するために何をすればよいか，なども具体化された．

5. 総　　括

検証例題以外にも，記述実験や実装実験から次のようなことが分かった．

5.1 Z 言語の採用について

1） 記述能力

状態遷移を記述するための言語として，記述能力はきわめて高い．今回の記述実験でも，仕様は必要十分に，しかも自己充足的に記述できた．

状態を記述するため，Z 言語は，集合論を包含している．そのためどのような状態（データ）であろうと記述できる．

状態遷移の記述は，Z 言語では，表明による．すなわち，対象となる状態の事前値と事後値および入力と出力が満たすべき条件を記述する方法だが，この方法はきわめて汎用性が高い．

2） 構造化記法

① 仕様部品を組み立てるための記述法と仕様部品を再利用するための記述法がよく整っている．このため

・記述の重複が避けられ，記述の効率化が図れる．

・記述の重複が避けられるので，仕様書を整合性を保ちながら保守できるようになる．

・巨大システムの仕様記述にも適用できそうだ，との見通しがえられた．

② Z 言語は，状態とその遷移を記述するための言語で，オブジェクト指向流の構造を明示するために必要な，状態と操作を局所化するための記述法がない．これをおぎなうため Z 言語の拡張版が種々考案されている(Object-Z その他)．

3) 支援ツール

① 型検査系の有効性

今回の実装実験では Z 言語の型検査系が必須であることを再認識した．

前述したように，共通例題 3 の仕様「リフトのシミュレーター」には誤りがあった．いくつかのスキーマを未定義のまま使用するというミスである．この種のミスは注意力ではなかなかカバーしきれない．だが，型検査系を使えばこの種のミスは直ちに検出できる．

Z 言語は強く型づけられた言語でる．しかもその型は極めて好ましい特性をもっている．それで書かれた仕様中の各表現の型が機械的に計算できる，という特性である．型検査系とは，Z 言語のこの特性を利用して，仕様中のすべての表現とそのすべての出現について，それを含む表現の中に潜む，型の不整合を検出するツールである．

このように，型検査系はきわめて好ましいツールだが，現存する型検査系は英語版だけで日本語版はまだない．日本語対応の型検査系を早急に開発すべきである．

② ZJ 翻訳系の有効性

実装者から受けた，Z 記法の意味を質す問いかけに，仕様作成者は，なるべく日常語に言い換えて説明している．もし Z 言語を日常の日本語に翻訳するツールがあれば，実装者はずいぶん助かるのではないか．翻訳ツールに対してなら，何度でも翻訳を依頼することができる．

このツールの恩恵を享けるのは実装時だけではない．このツールを使って，Z 仕様を日常語に翻訳すれば，その文書は，利用部門のための「システム説明文書」になる．説明文書が自動生成できるわけである．

4) コミュニケーションの手段

コミュニケーションのための手段として仕様言語は有効だろうか．仕様作成者も実装者も，実装実験を振り返って，Z 言語がコミュニケーションの手段として有効であったと指摘している．曰く

・実装者と仕様作成者間での質疑応答が，従来に比べて，少なかった

・主張の正否がハッキリするので議論が紛糾しない

・仕様理解にはとまどった面もあるが，理解してしまえばコーディングは即座にできた

と．

5.2 実 装 技 術

形式仕様によるコードの分散開発をなめらかに推進するためには，実装者にはどの

ような知識や技術が必要か.

1） 仕様言語理解

当然のことながら，仕様言語をよく理解する必要がある．単に知識として仕様言語を知るだけでなく,その言語で記述された仕様が実装できなければならない．ポイントは以下の 2 点（ 2)，3)）である．

2） データの実装技法

仕様言語で記述されたデータ型を実装言語でどう表現するのか，その定跡を豊富に知っていることが望ましい.

こんどの実装実験では，部分関数をテーブルで表現する手法などが，そうした定跡の一例である.

3） コードの構造化技法

① 仕様の構造とプログラムの構造

原則としては，コードの構造はあくまでアルゴリズムの都合で選ぶべきで，仕様の構造に左右されるべきものではない．だが，仕様の構造とプログラムの構造については，経験上，次のようなことが知られている.

多くの事務処理分野の課題では，課題の構造に即して仕様を構造化し，プログラムの構造もその構造に合わせるなら，分かりやすく仕様変更によく耐えるソフトウェアが構築できる，という経験則である.

そこで，実装者は，仕様の構造を実装言語によってどう表現するのか，そのための手法を知らなければならない.

② プログラミング技法

抽象データ型プログラミングやオブジェクト指向プログラミングなどのプログラミング手法がそのような定跡を提供してくれる.

③ アルゴリズム設計

ところで，今回の実装実験では，アルゴリズムの設計については議論されなかった．このことは,「事務処理分野のソフトウェア開発では，アルゴリズムの知識がなくても組めるようなプログラムが多い」せいだろう.

5.3 仕様言語教育

1） プログラミング＝仕様言語理解の近道

実装実験から,プログラミング課題に取り組むことが,仕様言語理解のために,どれほど有効であるか，このことを再認識した.

仕様言語教育で問題なのは，学習者がその言語をよく理解したかどうかが，判然としないことである．本当に理解したのかどうか，講師はもちろんのこと学習者本人ですらよく分からないことがある.

学習者が「自分はよく理解していない」と自覚できればまだよい．学習者は質問をすることができる．そして，それを契機に学習者と講師の間で質疑応答を繰り返すことによって，学習者は理解を深めることができるだろう．だが，この場合ですら，果たして本当に理解できたのか，この点はなんとも言えない.

しかし，正しいプログラムを作ることができれば，学習者がその仕様言語を理解できたことが判明する．正しいプログラムを作ることができなければ，仕様言

語がよく分かっていないことが判明する．

プログラミングとは仕様をプログラム言語に翻訳することである．だから，プログラミングに取り組めば，仕様のどこが翻訳できないか，つまりどの部分が分からないのか，分からない部分が特定できる．このため，学習者自ら分からない部分について質問できるようになる．やがて，その部分がプログラム言語に正しく翻訳できたとき，学習者はその部分を理解したことになる．プログラミング課題に取り組むこと，これこそ仕様言語理解のための最良の方法である．